CS 6923: Machine Learning

Fall 2018

Homework 4

Submit on NYU Classes by Mon. Nov 26 at 6:00 PM. You may work together with one other person on this homework. If you do that, hand in JUST ONE homework for the two of you, with both of your names on it. You may *discuss* this homework with other students but YOU MAY NOT SHARE WRITTEN ANSWERS OR CODE WITH ANYONE BUT YOUR PARTNER.

IMPORTANT SUBMISSION INSTRUCTIONS: Submit one file with the answers to the questions in Part I, another file with the answers to the written questions in Part II, and a third file with your code. Do NOT include the answers to part II questions in your code file.

1. In this exercise you will get practice with performing gradient descent in one variable.

   Consider the function

   $$f(x) = 2x^4 - 2x^3 - 12x^2 + 6$$

   Graph this function for x in the interval [-4,4]. You will see that it has two minima in that range, a local minimum and a global minimum. These are the only minima of the function.

   (a) What is the value of x at the local minimum and at the global minimum? Find the answer to this question like either by hand with calculus or using matlab or other software.

   (b) Suppose we apply gradient descent to this function, starting with $x = -4$. To do this, we will need to update x using the update rule
   $x = x + -\eta * f'(x)$
   where f' is the derivative of f, and η is the "step size".
   Write a small program implementating gradient descent for this function. Setting $x = -4$ and $\eta = 0.001$, run gradient descent for 6 iterations (that is, do the update 6 times). Report the values of x and $f(x)$ at the start and after each of the first 6 iterations.
   Run the gradient descent again, starting with x=-4, for 1200 iterations. Report the last 6 values of x and f(x).
   Has the value of x converged? Has the gradient descent found a minimum? Is it the global or the local minimum?
   You do NOT have to hand in your code.

   (c) Repeat the previous exercise, but this time, start with x=4.

   (d) Setting $x = -4$ and $\eta = 0.01$, run gradient descent for 1200 iterations. As in the previous two exercises, report the initial values of x and f(x), the next 6 values of x and f(x), and the last 6 values of x and f(x). Compare the results obtained this time to the results obtained above for $x = -4$ and $\eta = 0.001$. What happened?

   (e) Setting $x = -4$ and $\eta = 0.1$, run gradient descent for 100 iterations. What happened?

2. The backpropogation pseudocode we covered in lecture was for regression with 1 output node, using stochastic gradient descent.

   In some regression problems, we may want to predict more than one output value. For example, we might want to predict a person's location by specifying the longitude and latitude (2 numbers), or tomorrow's temperature and humidity. To use a single neural net for such problems, we add additional output nodes. Each of these output nodes is connected to the nodes in the previous layer.

   Pseudocode for backpropagation, training a neural net with K output nodes, is presented in Figure 11.11 in the textbook. This pseudocode assumes that the neural net has d input nodes (not counting $x_0 = 1$), H hidden nodes (not counting $z_0 = 1$), and K output nodes.

   Each hidden node h applies a sigmoid function to the weighted sum $w_h^T x$, where w_h is the vector of weights on the inputs to the node. So the output z_h of the node is equal to $sigmoid(w_h^T x) = \frac{1}{1+e^{-w_h^T x}}$. We say that the *activation function* at hidden node h is the sigmoid function. More generally, if z_h was equal to $g(w_h^T x)$ for some other function g, we would say that g was the activation function at hidden node h.

   Each output node just outputs the linear function $v_i^T z$ directly (so $y_i = v_i^T z$). This is equivalent to saying that the activation function at the output nodes is the identity function, since $y_i = g(v_i^T z)$ where g is the identity function (the function $g(t) = t$ for all $t \in \mathcal{R}$).

   The backpropogation performs gradient descent to minimize the sum of the squared errors, over all of the output nodes, on the input x^t:

   $E(W, v|\mathcal{X}) = \sum_i (\frac{1}{2}(r_i - y_i)^2)$

   (a) The pseudocode in Figure 11.11 is performing stochastic gradient descent. Each iteration of the repeat loop (processing all examples in the training set) is called an "epoch". Suppose there are 500 training examples each with 10 attributes, $K = 8$, and the neural net has 4 nodes in the hidden layer. If we run the above psuedocode for 100 epochs, how many times is weight v_{ih} updated, for $i = 2$ and $h = 1$?

   (b) The pseudocode in Figure 11.11 implements the gradient descent update rules for the weights, which are computed by taking partial derivatives. The value $\Delta(v_{ih})$ for updating the weight from hidden input number h to the ith output node is

$$\begin{aligned}
\Delta(v_{ih}) &= -\eta \frac{\partial E(W, v|\mathcal{X})}{\partial v_{ih}} \\
&= -\eta \frac{\partial}{\partial v_{ih}} (\frac{1}{2}(r_i - y_i)^2) \\
&= -\eta (r_i - y_i)(-\frac{\partial y_i}{\partial v_{ih}}) && \text{by the chain rule} \\
&= \eta (r_i - y_i) \frac{\partial}{\partial v_{ih}} (v_i^T z) && \text{by the definition of } y_i \\
&= \eta (r_i - y_i) z_h
\end{aligned}$$

   This is the LMS (least mean square) rule, if we focus just on output node i and the H inputs z_{ih} into it.

   Suppose that $K = 8$, $H = 4$ and instead of trying to minimize the sum of the squared errors of the outputs, you want to minimize a *weighted* sum of the squared errors of the outputs. Specifically, suppose you want to minimize the following new error function:

   $E_{new}(W, v|\mathcal{X}) = \frac{1}{2}[3(r_1 - y_1)^2 + 7(r_2 - y_2)^2]$

i. With the new error function, we need to modify the update rules. What is the value of $\Delta v_{2,3}$ ($i = 2$, $h = 3$) with the new error function?

ii. In the textbook, there is a derivation of the value $\Delta w_{h,j}$, which is used for updating the weight from input node j to hidden node h. Look at this derivation and then answer the following question: What is the value of $\Delta w_{h,j}$ for the new error function?

3. In the previous question, we considered a neural net for a regression problem with multiple outputs. In this question, we will also discuss neural nets for three other types of problems.

   To make it easier to refer to the four types of neural nets, we'll give them names. We'll call the neural net in the previous question NeuralNetRK (R for regression, K for the number of outputs). We now give the names of the other three types of neural nets, with their descriptions.

   NeuralNetCB: NeuralNetCB is a standard neural net for binary classification, with one hidden layer. It has a single output node which outputs a value y, where y is the predicted value of $P[Class1|x]$. It uses sigmoid functions as the activation functions both for the hidden nodes and for the output nodes. [1] The goal of the backpropagation is to minimize cross-entropy error. The pseudocode for backpropagation in this case is the same as in Figure 11.11, if d is set to 1, except that the line $y_i = v_i^T z$ has to be replaced by $y_i = \frac{1}{1+e^{-v_i^T z}}$.

   NeuralNetCK: This neural net is for classification with $K > 2$ classes. There are K output nodes i, and y_i is the predicted value of $P[Classi|x]$. In this case, each output y_i is equal to $\frac{e^{v_i^T z}}{\sum_{\ell=1}^{K} e^{v_\ell^T z}}$. Note that the sum of the y_i is 1, which is appropriate since the y_i are the estimated probabilities for the K labels of x. (In fact, the denominator in the expression for y_i is just a "normalizing factor" that causes the sum of the y_i to equal 1.) The error function is the generalization of cross-entropy to K classes: $-\sum_{i=1}^{K} r_i^t \log y_i^t$, where for each t, $r_i^t = 1$ if x^t is in Class i, and $r_i^t = 0$ otherwise. The pseudocode for backpropagation in this case is the same as in Figure 11.11, except that the line $y_i = v_i^T z$ has to be replaced by $y_i = \frac{e^{v_i^T z}}{\sum_{\ell=1}^{K} e^{v_\ell^T z}}$.

   NeuralNetRZeroOne: This neural net is designed for regression problems with K outputs where each output is either an element in the set $\{0, 1\}$, or a real value in the interval $[0, 1]$. The error function is squared error, the same as for NeuralNetRK. This neural net has sigmoid activation functions in both the hidden nodes AND the output nodes. So each output node computes $y_i = \frac{1}{1+e^{-v_i^T z}}$. Changing the pseudocode for backpropagation in this case requires changing both the line $y_i = v_i^T z$ and the lines computing Δv_i and Δw_h (as computed in bppy.py described below).

   (a) Which of the above neural nets can output values that are negative numbers?

   (b) Which of the above neural nets ensures that outputs $y_1, \ldots, y_K$ will satisfy $\sum_{i=1}^{K} = 1$?

   (c) Consider a simple image classification problem. Each image is a 10x10 array of pixels, and all pixel values are between 0 and 1. These pixel arrays are treated as example vectors of length 100, with one attribute per pixel. There are three categories: face, cat, and tree. These are represented by 3 output values, y_1, y_2, and y_3 where y_1 is 1 or 0 depending on whether or not the image is a face, y_2 is 1 or 0 depending on whether it is a cat, and y_3 is 1 or 0 depending on whether it is a tree.

   Suppose you want to use a neural net that will take an input image, and output three values p_1, p_2, p_3, where p_1 is the probability the image is a face, p_2 that it is a cat, and p_3 that it is a tree.

[1] Another popular choice of activation function for such a network is *tanh*, the hyperbolic tangent function. Deep nets often use ReLU (Rectified Linear Units) in the hidden nodes.

Which of the above neural nets would be appropriate for the image classification problem and why?

(d) Consider the following text classification problem. Each example is a document, represented by a binary vector of length n. Each attribute corresponds to a word, and the value is 1 or 0 depending on whether the word appears in the document. There are two outputs. The first is 1 or 0 depending on whether or not the document is about politics. The second is 1 or 0 depending on whether it is written in a formal or informal style.

Which of the above neural nets would be appropriate for the text classification problem and why?

4. Consider a classification problem where examples correspond to mushrooms, and the task is to determine whether the mushroom is edible (1) or poisonous (0) [2] Suppose one of the attributes is "odor" and it has the following 4 values:

- almond
- anise
- creosote
- fishy

There are actually two types of categorical variables (attributes): nominal and ordinal. The odor attribute is a nominal attribute, meaning the values (almond, anise, creosote, fishy) are not ordered.

The values of ordinal variables are ordered: e.g. if the values are low, medium, high, we would say that low < medium < high.

(Note: Sometimes in Machine Learning people talk about "nominal attributes" when they really mean "categorical attributes".)

(a) In order to use a neural net on this dataset, we need to decide how to convert the categorical values to numerical values. The obvious way to do this is to just assign a number to each value: almond (1), anise (2), creosote (3), fishy (4). This approach (directly converting attribute values to numbers) is sometimes called "label encoding".

This is NOT a good way to convert nominal values to numerical values, for use in a neural net. However, it would be fine to do this if we were using a random forest, rather than a neural net. Why?

(b) When converting nominal values to numerical values for neural nets (and a number of other learning methods), the following method is often used instead. It is often called one-hot encoding. In this method, for each possible value v of an attribute, we create a new binary-valued attribute whose value is 1 if the original attribute equals v, and 0 otherwise. We use these new attributes in place of the original attribute.

For example, we would replace the odor attribute above by 4 attributes we'll call z_1, z_2, z_3, z_4, where $z_1 = 1$ iff odor=almond, $z_2 = 1$ iff odor = anise, $z_3 = 1$ iff odor=creososte, and $z_4 = 1$ iff odor = fishy. If odor was the only attribute, and the original dataset was as follows

	odor	label
$x^{(1)}$	anise	0
$x^{(2)}$	creososte	1

then using one-hot encoding to convert the input attribute, the transformed dataset would be as follows:

[2]This question is inspired by the well-known mushroom dataset, https://archive.ics.uci.edu/ml/datasets/mushroom which was used in many Machine Learning experiments.

	z_1	z_2	z_3	z_4	label
$x^{(1)}$	0	1	0	0	0
$x^{(2)}$	0	0	1	0	1

i. Now suppose that the original dataset actually has two attributes, the odor attribute just described, and another attribute called *stalk shape*. The stalk-shape attribute has two possible values, tapering or enlarging.
What is the transformed dataset, if you apply one-hot encoding to the attributes in the following dataset?

	odor	stalk shape	label
$x^{(1)}$	fishy	tapering	0
$x^{(2)}$	creososte	enlarging	0

ii. Consider an ordinal attribute whose values are low, medium, and high. It might be better to represent these values as 1,2, and 3 rather than using one-hot-encoding. Why?

iii. The stalk shape attribute has only two values. For nominal attributes with only two values, it's generally fine to just represent the two values as 0 and 1 (or -1 and +1), rather than using one-hot encoding as described above. Why is this the case for attributes with 2 values, but not for attributes with more than two values?
(Notes: If the label values are categorical, we also need to convert them to numerical values for use in a neural net. Above, we converted “poisonous” and “edible” into 1 and 0.)

Part II: Programming Exercise

5. Consider again the movie review dataset that we used in HW2. Create a *vocabulary* consisting of all the tokens appearing in reviews in the training set. The initial attributes for this problem will be the tokens in the vocabulary. Each attribute will have the value 0 or 1, depending on whether the token appears or doesn't appear in the document. (Note: if a test document contains a token that is not in the vocabulary, you ignore it.)

   (a) List the 5 tokens that occur most frequently in the training set.

   (b) Using just the training documents, calculate the information gain of every attribute. List the 5 attributes with the highest information gain.

   (c) **Using only the 50 attributes with highest information gain**, train a neural net on the training examples. Your neural net should have one hidden layer with sigmoid units and a single output node. It should use the sigmoid function as the activation function in both the hidden units and the output unit, with cross-entropy as the loss function. You may modify the bppy.py code to do this, if you like.

   It is up to you to determine the correct learning rate and the number of hidden nodes. Do this using **only** on the training set.

   Then use the neural net with the learned weights (and the same set of attributes) to predict the outputs on the test set.

   Report your results on the test set using a confusion matrix. Also, list the percentage accuracy obtained.

   (d) What percentage accuracy is achieved if you use Zero-R, instead of a neural net?

   (e) We used only the top 50 attributes, but we could have used the top k attributes, for larger or smaller k. Why is it reasonable to think that increasing the number of attributes might increase accuracy? Why is it reasonable to think that decreasing the number of attributes might increase accuracy? Answer both questions. (then go to the next page for your final question)

(f) Perform experiments to see how accuracy changes as you vary the number k of attributes for your neural net. Choose at least 4 values of k (in addition to $k = 50$) and graph the results. The horizontal axis should correspond to the number of attributes, and the vertical to the test accuracy. Did the results surprise you? Did you have any difficulties running the experiments? Give the graph AND the answers to these two questions.